Remarks

Reconsideration of the application is respectfully requested in view of the following remarks. Claims 1, 3-7, 9, 10, 12-15, 17-24, and 26-32 are pending in the application. No claims have been allowed. Claims 1, 6, 12, 14, and 24 are independent. No claims have been amended.

Cited Art

The Office action ("Action") applies the following cited art: U.S. Patent No. 6,981,249 to Knoblock et al. ("Knoblock"); U.S. Patent No. 7,117,488 to Franz et al. ("Franz"); and U.S. Patent No. 6,560,774 to Gordon et al. ("Gordon").

§ 103 Rejection

The Action rejects claims 1, 3-7, 14, 17-23, and 29-31 under 35 U.S.C. § 103(a) as unpatentable over Knoblock in view of Gordon. The Action rejects claims 9, 10, 12-13, 15, 24, 26-28, and 32 under 35 U.S.C. § 103(a) as unpatentable over Knoblock in view of Gordon and in further view of Franz.

Response of Examiner's Arguments

In the prior response, Applicants argued that Knoblock does not teach or suggest rules for type-checking a type designated as an unknown type. In summary, Applicants pointed out that Knoblock describes a system for reconstructing types (using constraints) so that type checking can be performed using the reconstructed types.

In the Action, the Examiner argues that Knoblock's "constraints anticipate the rules for resolving unknown types." Action, page 3. Applicants agree. Knoblock uses a constraint collection technique to solve for unknown variable types so that type-checking can be performed. The difference is that Knoblock performs type-checking on reconstructed types (once types have been determined), not on unknown types.

In direct contrast to Knoblock, which performs type reconstruction so that type-checking can be performed, the claims generally recite rules for type-checking unknown types. For example, claim 1 recites (emphasis added):

type-checking the one or more representations based on a rule set, wherein the **rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known**.

Applicants provide further description of Knoblock and individual claim language in the sections below.

Claim 1

Claim 1 recites (emphasis added):

> A method of type-checking a code segment written in a programming language comprising:
> translating the code segment from the programming language to one or more representations of an intermediate language, *wherein the one or more representations of the intermediate language are capable of representing programs written in a plurality of different source languages, wherein the plurality of different source languages comprise at least one typed source language and at least one untyped source language*; and
> *type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known.*

Knoblock's description of reconstructing types does not teach or suggest "wherein the one or more representations of the intermediate language are capable of representing programs written in a plurality of different source languages, wherein the plurality of different source languages comprise at least one typed source language and at least one untyped source language" or "type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known," as recited by claim 1.

Knoblock is directed to type reconstruction. As Knoblock describes, when a source program is translated (e.g., compiled) to produce a bytecode program, some types that were present in the source program are lost. Knoblock, col. 5, lines 59-66. Knoblock states that before type checking can begin, a program must have types, "But to even begin the process of type checking a program, the program has to have types." Knoblock, col. 5, lines 58-59. Therefore, Knoblock provides a solution for reconstructing types that were lost during translation of a source program to a bytecode program. Knoblock, col. 5, line 58 to col. 6, line 13. The solution described by Knoblock for reconstructing types involves a constraint collection technique. Specifically, Knoblock states, "the embodiments of the present invention provide a constraint collection technique to learn from the remaining portions of the method 320 to solve for the type of the local variable x." Knoblock, col. 7, lines 21-31. Knoblock describes one method for reconstructing types involving labeling variables as unknown, collecting constraints between known types and unknown types, and solving for the unknown types using the constraints. Knoblock, col. 8, lines 4-55.

Knoblock's does not teach or suggest "*wherein the one or more representations of the intermediate language are capable of representing programs written in a plurality of different source languages, wherein the plurality of different source languages comprise at least one typed source language and at least one untyped source language*" as recited by claim 1. For example, see the Application at page 3, lines 5-14 and page 5, lines 22-27. Instead, Knoblock generally describes "a program that is written in a computer programming language, such as Java." Knoblock col. 5, lines 32-35.

The Examiner acknowledges that Knoblock does not disclose this claim language, but argues that Gordon does. Specifically, the Examiner argues that Gordon describes "at least one untyped source language (see at least Lisp, Scheme, Smalltalk col. 35:8-20)." Applicants respectfully disagree. Gordon explicitly states, at col. 35, lines 8-20, that these source languages (Lisp, Scheme, and Smalltalk) are dynamically typed languages, not untyped source languages, as follows (emphasis added):

> By-ref parameters and value classes are sufficient to support statically typed languages (Java, C++, Pascal, etc.). They also support <u>dynamically typed languages</u> that pay a performance penalty to box value classes before passing them to polymorphic methods (Lisp, Scheme, SmallTalk, etc.).

Instead of statically typed and dynamically typed languages, as described by Gordon, claim 1 recites, "wherein the one or more representations of the intermediate language are capable of representing programs written in a plurality of different source languages, wherein the plurality of different source languages comprise at least one typed source language and at least one untyped source language." Gordon never even mentions an "untyped" source language.

Furthermore, Knoblock does not teach or suggest "*type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type* *designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known*" as recited by claim 1. The Examiner argues that Knoblock describes this language. Action, page 5. Applicants respectfully disagree. Knoblock does not describe rules for type checking a type desiganted as an unknown type. In fact, Knoblock reconstructs types in order to perform type-checking on the reconstructed types, as described above. In addition, as understood by Applicants, Gordon does not teach or suggest the above-cited language of claim 1.

For at least these reasons, Knoblock, separately or in combination with Gordon, does not each or suggest the language of claim 1. Therefore, claim 1 should be in condition for allowance.

Claim 6

For at least the reasons stated above with regard to claim 1, Knoblock, separately or in combination with Gordon, does not teach or suggest "type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking the type designated as the unknown type," as recited by claim 6. Therefore, claim 6 should be in condition for allowance.

Claim 14

For at least the reasons stated above with regard to claim 1, Knoblock, separately or in combination with Gordon, does not teach or suggest "one or more rule sets comprising rules associated with the type, designated as the unknown type, indicating an element can be of any type; and a type-checker for applying the one or more rule sets to the elements of the intermediate representation," as recited by claim 14. Therefore, claim 14 should be in condition for allowance.

Claims 3-5, 7, 17-23, 29-31

Claims 3-5, 29, and 30 depend on claim 1. Thus, for at least the reasons set forth above with regard to claim 1, claims 3-5, 29, and 30 should be in condition for allowance.

Claims 7 and 31 ultimately depend on claim 6. Thus, for at least the reasons set forth above with regard to claim 6, claims 7 and 31 should be in condition for allowance.

Claims 17-23 ultimately depend on claim 14. Thus, for at least the reasons set forth above with regard to claim 14, claims 17-23 should be in condition for allowance.

Claims 12 and 24

Claim 12 recites (emphasis added):

> replacing the types associated with the plurality of programming languages with the types of the intermediate language, wherein the types of the intermediate language comprise general categories of the types associated with the plurality of programming languages and a type designated as an unknown type, *wherein the type designated as the unknown type has size information associated with it, wherein the size information comprises size information of a machine representation of the type designated as the unknown type.*

Claim 24 recites (emphasis added):

> *wherein the type indicating that an element of the intermediate language is associated with the type designated as the unknown type has a size associated with it, wherein the size represents size of a machine representation of the type designated as the unknown type.*

The Examiner argues that Franz teaches the above-cited language of claims 12 and 24 respectively, citing Franz description of array size at col. 11, line 63 to col. 12, line 11. Applicants respectfully disagree. Franz describes size with relation to array indexing (bounds), so that array indexes are safe, “Similarly, for an array arr, a matching type safe-index-arr is provided, whose instances may assume only values that are index values within the legal range defined for the array arr.” Franz, col. 11, line 63 to col. 12, line 11. Furthermore, Franz states, “The size of an array may not be known statically, but once the array object has been created, its size will remain constant.” Franz, col. 12, lines 5-7. Franz describes the need for enforcing array index checking at col. 1, line 60 to col. 2, line 3:

> Assume an array is declared as having ten elements. A malicious code provider might wish to access array element eleven, thereby circumventing the type rules and gaining access to whatever variable happened to be located at the memory location corresponding to element eleven of the array--even if that variable is marked as being private or protected. Many exploits of security holes use this route, using a breach of type safety to modify variables that they normally would not have access to. Type safe code prevents this by disallowing the referencing of array elements beyond those defined to be in the array.

As described by Franz, array indexes may be made safe. Checking array index bounds does not require a machine representation size of array element types. For at least these reasons, Franz does not teach or suggest that a type designated as an unknown type with associated machine-representation size information associated with the unknown type. Furthermore, as understood by Applicants, Gordon does not add sufficient disclosure to teach or suggest the above-cited language of claims 12 and 24.

Knoblock separately or in combination with Franz and/or Gordon, does not teach or suggest the above-cited language of claims 12 and 24, respectively. Therefore, claims 12 and 24 should be in condition for allowance.

Claims 9, 10, 13, 15, 26-28, and 32

Claims 9 and 10 ultimately depend from claim 6, claims 15 and 32 ultimately depend from claim 14. Therefore, for at least the reasons stated above with regard to claims 6 and 14, Knoblock does not teach or suggest the language of claims 9, 10, 15, and 32 (as dependent on their respective independent claims) cited above. Furthermore, as understood by Applicants, Franz and/or Gordon do not add sufficient disclosure to teach or suggest this language. Thus, claims 9, 10, 15, and 32 should be in condition for allowance.

Claim 13 depends on claim 12. Thus, for at least the reasons set forth above with regard to claim 12, claim 13 should be in condition for allowance.

Claims 26-28 depend on claim 24. Thus, for at least the reasons set forth above with regard to claim 24, claims 26-28 should be in condition for allowance.

Request for Interview

If any issues remain, the Examiner is formally requested to contact the undersigned attorney prior to issuance of the next Office action in order to arrange a telephonic interview. It is believed that a brief discussion of the merits of the present application may expedite prosecution. Applicants submit the foregoing remarks so that the Examiner may fully evaluate Applicants' position, thereby enabling the interview to be more focused.

This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

Conclusion

The claims should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By /Cory A. Jones/
Cory A. Jones
Registration No. 55,307